Computing and Informatics, Vol. 30, 2011, 491–511

A PERSONALIZED FACET-WEIGHT BASED RANKING METHOD FOR SERVICE COMPONENT RETRIEVAL

Ming Zhong, Yaoxue Zhang

Tsinghua National Laboratory for Information Science and Technology
Beijing, 100084, China
&
Department of Computer Science and Technology
Tsinghua University
Beijing, 100084, China
e-mail: zhong-m@mails.tsinghua.edu.cn

Laurence Tianruo Yang

Department of Computer Science
St. Francis Xavier University
Antigonish, NS, B2G 2W5, Canada

Yuezhi Zhou, Pengwei Tian, Linkai Weng

Tsinghua National Laboratory for Information Science and Technology
Beijing, 100084, China
&
Department of Computer Science and Technology
Tsinghua University
Beijing, 100084, China

Abstract. With the recent advanced computing, networking technologies and embedded systems, the computing paradigm has switched from mainframe and desktop computing to ubiquitous computing, one of whose visions is to provide intelligent, personalized and comprehensive services to users. As a new paradigm, Active Services is proposed to generate such services by retrieving, adapting, and composing of existing service components to satisfy user requirements. As the popularity of this paradigm and hence the number of service components increases, how to efficiently retrieve components to maximally meet user requirements has become a fundamental and significant problem. However, traditional facet-based retrieval methods only simply list out all the results without any kind of ranking and do not lay any emphasis on the differences of importance on each facet value in user requirements, which makes it hard for user to quickly select suitable components from the resulting list. To solve the problems, this paper proposes a novel personalized facet-weight based ranking method for service component retrieval, which assigns a weight for each facet to distinguish the importance of the facets, and constructs a personalized model to automatically calculate facet-weights for users according to their historical retrieval records of the facet values and the weight setting. We optimize the parameters of the personalized model, evaluate the performance of the proposed retrieval method, and compare with the traditional facet-based matching methods. The experimental results show promising results in terms of retrieval accuracy and execution time.

Keywords: Active services, component ranking, facet-weight, ubiquitous computing

1 INTRODUCTION

With the rapid development of software, hardware and computer network technologies, the computing paradigm has been changing radically in the recent 20 years. It is a certain trend that the ubiquitous computing is replacing the mainframe and desktop computing. Ubiquitous computing lays the emphasis on the seamless integration of human, computers as well as environment, and aims at providing ubiquitous services to meet users' intelligent, personalized and comprehensive requirements anytime, anywhere and by any means. However, existing services and computing paradigms are not flexible and powerful enough to meet users' various requirements. How to customize services dynamically according to users' requirements has become a hot research field in ubiquitous computing.

Active Services [1], as a new paradigm for ubiquitous service customization, has been proposed recently to solve the above-mentioned problems and has achieved popularity in today's software development community due to the increasing complexity of services, increasing costs of maintenance, and decreasing costs of underlying hardware. It encapsulates existing resources such as services, programs and devices into reusable service components. In the following parts of the paper, for simplification purpose, we use component to denote service component. The paradigm of Active Services makes use of these components distributed in the ubiquitous environment, and generates new services to meet users' personalized requirements with the technology of Component-Based Software Development (CBSD) [2]. Therefore, the process of developing ubiquitous services has changed to the combination of existing components through retrieval, adaptation, composition of the components, and testing of the developed services. Since all of these activities are influenced by component retrieval, it becomes a critical process for service customization. In order to keep high efficiency for generating ubiquitous services to satisfy users' various requirements, it is really important to retrieve the most suitable component quickly and accurately among a large-scaled components repository.

There are extensive retrieval methods used for component repositories [3, 4, 5, 6, 7, 8]. Among them, the facet-based component retrieval method has been proved to be an effective way for retrieving. The facet classification in component repository was proposed by Prieto-Diaz and Freeman in 1987 [9]. A faceted scheme consists of a group of facets that describe the essential features of components in an objective and comprehensive way. Each facet classifies components from different angles, and consists of a set of terms, called term space. The process of the facet-based component retrieval method is to match each facet between user queries and components. It has been widely applied by various component reuse organizations, such as NATO [10] and REBOOT [11]. Most of these organizations use the traditional database query techniques in facet-based component repository. In order to improve the efficiency of facet-based component retrieval method, many researches have been taken [12, 13, 14]. Gibb [15] used XML to describe component facets and applied XML-SQL as the searching language to achieve component retrieval in their project. In [16], Wang proposed a tree matching algorithm for facet-based component repository, which maps the component facets into a facet tree and the query conditions into a query tree. These methods have improved and expanded the facet-based component retrieval method by adopting efficient structure of facet scheme. However, they still have two disadvantages:

1. these retrieval methods usually simply list out all the results without any kind of ranking;
2. user requirements can only be expressed by the selected values on each facet, but the differences of importance on each facet value are ignored.

These two problems lead to the scenario that users have to view the detailed information of all the listed components to determine which one satisfies their requirements, which is of low efficiency in component retrieval. Then in [17], Yang set weight for six fixed facets and proposed a mathematical model of weighted ranking algorithm. Furthermore, Xie [18] designed an intelligent component retrieval model – FWRM, which changes the facet-weight itself dynamically by genetic algorithm, and uses risk minimization-based component sampling method to solve the insufficiency of training data. These two methods solve the ranking problem of component retrieval, and give efficient ways to calculate the facet weight; but they do not consider in calculating the facet-weight the differences of user's private retrieval habits, interests and understanding of the facet terms, which is far away from the target of providing ubiquitous and on-demand services with high efficiency. So it is important and necessary to distinguish the importance of each facet value for each user and to define a formula to calculate the matching degree to rank the components in the resulting list.

In this paper, we propose a personalized facet-weight based ranking method for component retrieval. We formally specify the components and user queries under facet classification space in the form of vectors, and assign a facet-weight on each facet for a user to decide which facet value in his query is more important. The user's setting of facet-weights provides more information of his requirement, and helps rank the components with finer granularity. Furthermore, the user's setting of facet-weight is proved to be related to his query on each facet, so it is possible to extract user's retrieval habits and interests from his historical retrieval sequence of facet values and facet-weights. According to the historical records, we construct a personalized model. The model is used to automatically calculate weights for users when it brings forward new retrieval facet values, so that the user will save much time in facet-weight setting. The experimental results optimize the parameters of the personalized model for the best facet-weight vector, and prove that the personalized facet-weight based component retrieval method performs better in retrieval accuracy and takes shorter time to find the target component.

The rest of this paper is organized as follows. Section 2 defines the facet vectors and the facet matching degree, introduces the facet-weight based component retrieval method, and then gives an example. Section 3 describes the personalized model and the facet-weight calculation based on the model. We describe our experimental results and analysis in Section 4 and conclude the work in Section 5.

2 COMPONENT RANKING METHOD BASED ON FACET-WEIGHT

In this section we describe the definitions of the facet vectors for components and user queries, as well as the definitions of the facet matching degree and the facet-weight. Then we introduce the ranking formula based on facet-weight for component retrieval.

2.1 Facet Vector and Facet Matching Degree

The key idea of facet classification is to accurately classify components so as to reflect the essence of components. Current component repositories usually employ a group of facets to classify their components. Prieto-Diaz and Freeman [9] take *Function, Object* and *Medium* as facets in their system, while the Jade Bird Component Library System [19] contains the facets of *Application Domain, Component Function, Component Type, Programming Language* and *Running Platform.* Generally, we assume that there are n facets to describe a component, and each facet contains a limited set of facet terms.

Definition 1 (Facet Vector)**.** The facet-based structure of component repository is defined as a vector $\vec{F} = (\vec{f_1}, \vec{f_2}, \ldots, \vec{f_n})$ in which $\vec{f_i} = (f_{i,1}, f_{i,2}, \ldots, f_{i,n_i})$. $\vec{f_i}$ is the

i^{th} facet to describe component, n_i is the number of facet values under $\vec{f_i}$, and $f_{i,j}$ represents one of the facet terms under $\vec{f_i}$.

Generally, the structure of facet-based component repository is tree-styled and hiberarchical. However, in the construction of the *Facet Vector*, we only choose the terms of lowest level of a facet, because these terms are enough to reflect the characters of components and user requirements.

The values on each facet of a component are described by component providers, according to the properties of the component and the facet space of the target repository. So the component can be described as *Component Vector*, based on *Facet Vector*.

Definition 2 (Component Vector)**.** Under the vector space of facet $\vec{F}$, each component can be defined as $\vec{c} = (\vec{c_1}, \vec{c_2}, \ldots, \vec{c_n})$, in which $\vec{c_i}$ is a vector generated by all the facet values under the i^{th} facet $\vec{f_i}$, defined as follows:

$$\vec{c_i} = (c_{i,1}, c_{i,2}, \ldots, c_{i,n_i}), c_{i,j} = \begin{cases} 1 & \text{if } \vec{c} \text{ related to } f_{i,j} \\ 0 & \text{otherwise} \end{cases} \quad j = 1, 2, \ldots, n_i$$

For example, suppose that there are 5 facet values under the facet of running platform: *winXP, win2k, linux, Unix, others.* Then the facet value vector of this facet should be defined as $\vec{f_i} = (winXP, win2k, linux, Unix, others)$. If a component is supposed to run on *linux*, its vector of this facet is $(0, 0, 1, 0, 0)$; and if a component is supposed to run on *winXP* and *win2k*, its vector is obviously $(1, 1, 0, 0, 0)$.

A user query is brought forward on the foundation of facet classification scheme of the component repository; so it can also be formally defined based on *Facet Vector*, same as the structure of *Component Vector.*

Definition 3 (User Query Vector)**.** Under the vector space of facet $\vec{F}$, each user query can be defined as $\vec{q_0} = (\vec{q}_{0,1}, \vec{q}_{0,2}, \ldots, \vec{q}_{0,n})$, in which $\vec{q}_{0,i}$ is a vector generated by all the user required values on the i^{th} facet $\vec{f_i}$, defined as follows:

$$\vec{q}_{0,i} = (q_{0,i,1}, q_{0,i,2}, \ldots, q_{0,i,n_i}), q_{0,i,j} = \begin{cases} 1 & \text{if user selects } f_{i,j} \\ 0 & \text{otherwise.} \end{cases} \quad j = 1, 2, \ldots, n_i$$

Based on the definition of *Component Vector* and *User Query Vector*, the *Facet Matching Degree (FMD)* indicates how similar two vectors are under a facet. It reflects the similarity and relevancy between a component and a user query, or two user queries. As both the *Component Vector* and *User Query Vector* are following the same structure, we use the inner product to calculate the *FMD.* Assuming that $\vec{v}_{i,j}$ and $\vec{v}_{k,j}$ are two vectors from the j^{th} facet of $\vec{v_i}$ and $\vec{v_k}$, then the *FMD* of $\vec{v}_{i,j}$ and $\vec{v}_{k,j}$ is defined as.

$$FMD(\vec{v}_{i,j}, \vec{v}_{k,j}) = \vec{v}_{i,j} \cdot \vec{v}_{k,j} = \sum_{l=1}^{n_j} v_{i,j,l} \cdot v_{k,j,l}. \tag{1}$$

2.2 Component Ranking Formula

On the basis of the *Facet Vector* based definitions of components and user queries, the *Facet Matching Degree* indicates the similarity on each facet; but if the *FMD* of each facet is directly and simply added together, the difference of importance of those facets can not be distinguished. In the process of component retrieval, different users usually emphasize different facets in their requirements, according to the requirement content, personal habits, etc. For example, when selecting a component for calculator of java applet, users may think the matching on facet of *Component Function* and *Component Type* is more important than that of *Application Domain.* So it is necessary to set a weight for each facet to differentiate the user's view on the facet values. Users are able to define the weights of each facet themselves, according to what their requirements really are.

As it is supposed there are n facets to describe components, the number of weight in a facet-weight vector should also be n. It is defined as follows.

Definition 4 (Facet-Weight)**.** A *Facet-Weight* vector is defined as $\vec{w_0} = (w_{0,1}, w_{0,2}, \ldots, w_{0,n})$. $w_{0,i}$ represents the weight of *Facet Matching Degree* on the i^{th} facet $\vec{f_i}$. The vector $\vec{w_0}$ is normalized such that:

$$\sum_{i=1}^{n} {w_{0,i}}^2 = 1.$$

Then the *General Matching Degree (GMD)* between user query $\vec{q_0}$ and component $\vec{c}$ is defined as the weighted sum of matching degrees on all facets:

$$GMD(\vec{q_0}, \vec{c}) = \sum_{i=1}^{n} FMD(\vec{q_{0,i}}, \vec{c_i}) \cdot w_{0,i}. \tag{2}$$

Equation (2) reflects how much the component $\vec{c}$ satisfies the user query $\vec{q_0}$. The larger the resulting number is, the closer is the relationship between them, especially on the facets with high weight. So we can calculate the *GMD* between $\vec{q_0}$ and each component $\vec{c}$ in the repository, and sort components by their *GMDs*.

An example is presented in Figure 1 to demonstrate the process of component ranking based on facet-weight and its advantage comparing with traditional facet-based matching method. The user has proposed facet values on five facets as his query, and the two different components satisfy the most part of the user query. Component 1 satisfies the user query except "*Book hotel*" and "*View map*" on *Component Function* facet, while component 2 satisfies user query except "*Book hotel*" on *Component Function* facet and "*ActiveX EXE*" on *Component Type* facet. We can calculate the *FMD* on each facet between user query and component. If the *FMD* of each facet is simply added together, the *GMD* between the user query and the two components are the same. Without the information of which facets the user emphasizes, traditional facet matching method can not distinguish these two components; but after the user set a weight for each facet as $(0.8, 0.3, 0.3, 0.3, 0.3)$,

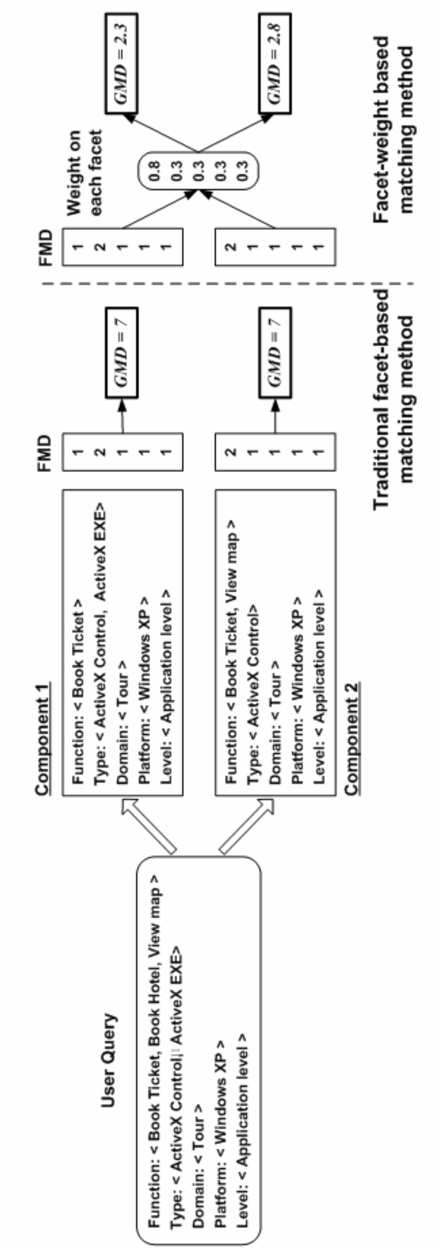
User Query
Function: < Book Ticket, Book Hotel, View map >
Type: < ActiveX Control, ActiveX EXE>
Domain: < Tour >
Platform: < Windows XP >
Level: < Application level >
Component 1
Function: < Book Ticket >
Type: < ActiveX Control, ActiveX EXE>
Domain: < Tour >
Platform: < Windows XP >
Level: < Application level >
Component 2
Function: < Book Ticket, View map >
Type: < ActiveX Control>
Domain: < Tour >
Platform: < Windows XP >
Level: < Application level >
FMD
1 2 1 1 1
2 1 1 1 1
GMD = 7
GMD = 7
Traditional facet-based matching method
FMD
1 2 1 1 1
2 1 1 1 1
Weight on each facet
0.8 0.3 0.3 0.3 0.3
GMD = 2.3
GMD = 2.8
Facet-weight based matching method

Fig. 1. Comparing example between facet matching and facet-weight matching methods

it is clear that the user indicates the matching of *Component Function* facet is the most important in his current requirement. According to Equation (2), we can figure out that the *GMD* of Component 1 is 2.3, smaller than the *GMD* of component 2, which is 2.8. Actually we can see that component 2 satisfies more user needs, and it should be laid in front of component 1 in the retrieval result.

3 PERSONALIZED FACET-WEIGHT CALCULATION

There are two kinds of significant information which influence the setting of facet-weight in the process of component retrieval: user requirements and user customs. So generally, if two retrieval queries from a user are similar, he will usually set similar facet-weight vectors on them. For example, a user searching for a video player component will usually focus on matching of the *Component Function* facet, while if he needs a chatting tool component, he is likely to emphasize matching of the *Component Type* facet. Therefore, it is useful to capture how a user's retrieval history is related to his weight setting, and create a personalized model to identify and learn his interests and habits on component retrieval. The personalized model helps calculate weight of each facet, according to user's current query. It could efficiently improve accuracy in generating and ranking the retrieval results.

3.1 Personalized Modeling and Facet-Weight Calculation

In our component retrieval system, constructing user's personalized model is used to calculate facet-weights for the user's current query. To achieve the goal, it is necessary to collect the user's historical information of component retrieval, and extract the user's common interests and habits on setting facet-weights for his query. So here we define two kinds of information to create the user personalized model for facet-weight calculation.

1. **User's historical retrieval records.** User's historical operations of component retrieval directly contain the information of user interests and habits, especially the relationship between the query vector and the corresponding facet-weight vector. So we define a retrieval log for each user, restoring his historical retrieval records. However, not all of the user's historical retrieval records are inserted into the log. If the component user wants is not in top 10 of the resulting list, this retrieval operation will be regarded as an invalid one and will not be restored in the log, because there must be something wrong in the setting of user query or facet-weights, and wrong records may lead to failure in personalized facet-weight calculation. Here each record item in the log is defined as $r_i = (\vec{q_i}, \vec{w_i})$ in which $\vec{q_i} = (\vec{q}_{i,1}, \vec{q}_{i,2}, \dots, \vec{q}_{i,n})$ is the user query vector of the i^{th} record item, and $\vec{w_i} = (w_{i,1}, w_{i,2}, \dots, w_{i,n})$ is the facet-weight vector of the i^{th} record item.
2. **Fading value.** Each record in the user retrieval log is sorted by its access time. New records are put behind the old ones. User's interests and habits will change with time and user's latter query and weights setting are better than earlier ones

in reflecting user's interests and habits. So we define a fading factor ϵ, which is a constant parameter and $0 < \epsilon \leq 1$. Based on the fading factor, facet value for the i^{th} record is defined as $s_i = \epsilon^{m-i}$, in which m is assumed as the number of records. When $i = m, s_i = s_m = 1$ means the latest record needs no fading; and when $i = 1, s_i = s_{m-1} = \epsilon^{m-1}$ means the oldest record gets the smallest fading value.

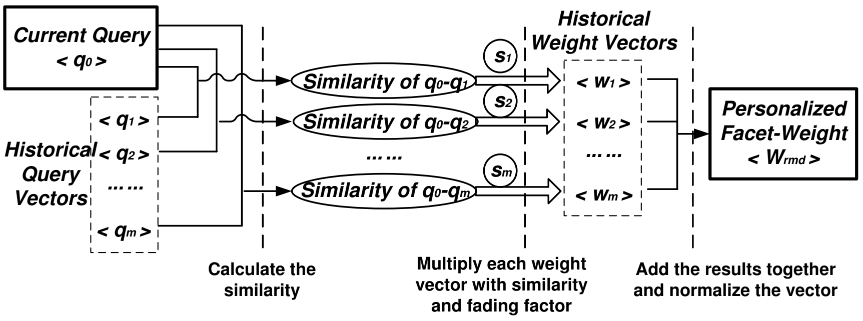

Fig. 2. Process of personalized facet-weight calculation based on historical retrieval records

The key idea behind personalized facet-weight calculation is to highlight user's weights setting of similar historical queries and recent retrieval records. We still assume that the user's new retrieval query is $\vec{q_0} = (\vec{q}_{0,1}, \vec{q}_{0,2}, \ldots, \vec{q}_{0,n})$ and the number of user's historical records is m. Figure 2 shows the process of how to calculate the personalized facet-weight on the basis of historical retrieval records. It can be divided into three steps:

Step 1: Calculate the similarity between user's current query and each record in the retrieval log, based on the definition of *FMD*. If one of the user's retrieval queries in the past is much closer to the user's current query, its corresponding facet-weights setting on each facet will certainly be of higher reference value.

Step 2: Change the facet-weight vector of each record by multiplying the corresponding similarity calculated in (1) and the fading value into every element of the facet-weight vector.

Step 3: Add all the updated facet-weight vectors from (2) together, and normalize the resulting vector to calculate the final facet-weight vector. Therefore, the calculating formula of personalized facet-weight $\vec{w}_{rmd} = (w_{rmd-1}, w_{rmd-2}, \ldots, w_{rmd-n})$ is described as follows:

$$\vec{w}_{rmd} = \sum_{i=1}^{m} \left(\sum_{j=1}^{n} FMD\left(\vec{q}_{0,j}, \vec{q}_{i,j}\right) \right) \cdot s_i \cdot \vec{w}_i. \tag{3}$$

The final facet-weight vector is not normalized and will not influence the order of retrieval results. However, it is still necessary to normalize the facet-weight vector before inserting it as well as the user query vector into the user retrieval log, and the normalized facet-weight vectors are easy for users to understand and adjust.

Equation (3) can be regarded as a raw model for facet-weight calculation. The process of facet-weight calculation requires comparing and integrating the current user query with all the historical queries and facet-weights. With the accumulation of log records over time, the storage spending will be exhausted and the computational complexity will significantly increase; so it is necessary to make the model independent with the number of historical records, while reserving the essential characters on the user's setting of queries and weights.

Here we derive the raw model based on Equation (3), and extract a user personalized model for facet-weight calculation. The t^{th} weight of the facet-weight vector, assigned as w_{rmd-t}, can be calculated and deduced as follows:

$$\begin{aligned}
\vec{w}_{rmd-t} &= \sum_{i=1}^{m}\left(\sum_{j=1}^{n} FMD\left(\vec{q}_{0,j}, \vec{q}_{i,j}\right)\right)\cdot s_i \cdot w_{i,t} \\
&= \sum_{i=1}^{m}\left(\sum_{j=1}^{n}\left(\sum_{l=1}^{n_j} q_{0,j,l}\cdot q_{i,j,l}\right)\right)\cdot s_i \cdot w_{i,t} \\
&= \sum_{i=1}^{m}\left(\sum_{j=1}^{n}\left(\sum_{l=1}^{n_j} q_{0,j,l}\cdot q_{i,j,l}\cdot w_{i,t}\right)\right)\cdot s_i \\
&= \sum_{j=1}^{n}\left(\sum_{l=1}^{n_j}\left(\sum_{i=1}^{m} q_{0,j,l}\cdot q_{i,j,l}\cdot w_{i,t}\cdot s_i\right)\right) \\
&= \sum_{j=1}^{n}\left(\sum_{l=1}^{n_j} q_{0,j,l}\cdot\left(\sum_{i=1}^{m} q_{i,j,l}\cdot w_{i,t}\cdot s_i\right)\right) \\
&= \sum_{j=1}^{n}\left(\vec{q}_{0,j}\cdot\left(\sum_{i=1}^{m}\vec{q}_{i,j}\cdot w_{i,t}\cdot s_i\right)\right).
\end{aligned}$$

Define a new vector $\vec{u_t} = (\vec{u}_{t,1}, \vec{u}_{t,2}, \ldots, \vec{u}_{t,n})$ in which:

$$\vec{u}_{t,j} = \sum_{i=1}^{m}\vec{q}_{i,j}\cdot w_{i,t}\cdot s_i. \tag{4}$$

Thus, we have the following modified formula:

$$w_{rmd-t} = \sum_{j=1}^{n}\left(\vec{q}_{0,j}\cdot\left(\sum_{i=1}^{m}\vec{q}_{i,j}\cdot w_{i,t}\cdot s_i\right)\right) = \sum_{j=1}^{n}\left(\vec{q}_{0,j}\cdot\vec{u}_{t,j}\right). \tag{5}$$

According to the deduction above, we extract the personalized model $U = \{\vec{u_t}, t = 1, 2, \ldots, n\}$ in which the t^{th} element $\vec{u_t}$ is a vector independent in calculating the weight of the t^{th} facet. Figure 3 shows the process of personalized facet-weight calculation after personalized model extraction. We can see that the personalized model U has no relationship with the number of records in user retrieval log. So after the process of modeling, the used records could be deleted from log, so as to save the storage spending for user retrieval log; and the computational complexity of facet-weight is greatly reduced from $O(mn\sigma)$ to $O(n\sigma)$, in which σ is the total number of all facet terms, n and m are still the number of facets and the number of user's historical retrieval records, respectively.

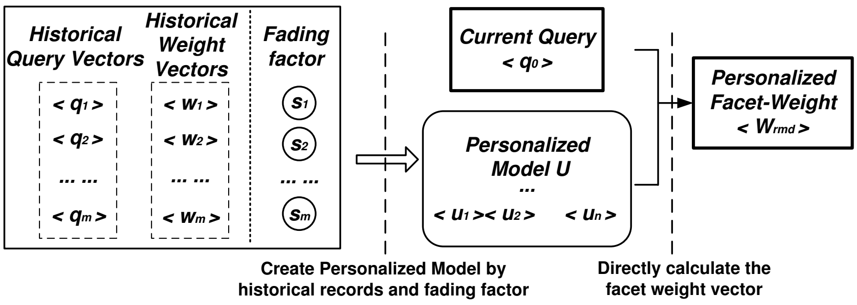
Historical Query Vectors
Historical Weight Vectors
Fading factor
< q1 >
< q2 >
... ...
< qm >
< w1 >
< w2 >
... ...
< wm >
s1
s2
... ...
sm
Current Query < q0 >
Personalized Model U
...
< u1 >< u2 > < un >
Personalized Facet-Weight < Wrmd >
Create Personalized Model by historical records and fading factor
Directly calculate the facet weight vector

Fig. 3. Process of personalized facet-weight calculation after personalized model extraction

3.2 Incremental Update

A user's personalized model U is not constant, because the user keeps on searching components and new records of facet values and weights are expected to be updated and inserted, which are more important in reflecting user's interests and habits. So it is necessary to incrementally update the user's personalized model by inserting user's new queries and weights setting into the model. The incremental one or more records have the same structure as the ones that are taken to construct the personalized model U. So we execute the same steps to construct the model's incremental part U' and add U' to the original model U to achieve the user's personalized model update. Incremental update process includes two steps:

Step 1: Generally, assuming that the original personalized model U is constructed based on m historical retrieval records, $R = \{r_1, r_2, \ldots, r_m\}$ and p new records, $R' = \{r_{m+1}, r_{m+2}, \ldots, r_{m+p}\}$ are going to be added into the model. According to the newer p records, we then construct the incremental part $U' = \{\vec{u'_t}, t = 1, 2, \ldots, n\}$ in which the jth element of $\vec{u'_t}$ is

$$\vec{u'}_{t,j} = \sum_{i=m+1}^{m+p} \vec{q_{i,j}} \cdot w_{i,t} \cdot s_i. \tag{6}$$

As the number of historical records has been changed to $m + p$, the formula of fading factor should be also changed to $s_i = \epsilon^{m+p-i}$.

Step 2: Integrate each vector $\vec{u_t'}$ of U' to its corresponding vector $\vec{u_t}$ of original model U with vector addition operation, which results in $U_{new} = \{\vec{u}_{new-t}, t = 1, 2, \ldots, n\}$. The j^{th} element of $\vec{u}_{new-t}$ in U_{new} can be deduced and proved as follows:

$$\begin{aligned}
\vec{u}_{mew-t,j} &= \sum_{i=1}^{m+p} \vec{q}_{i,j} \cdot w_{i,t} \cdot s_i \\
&= \sum_{i=1}^{m} \vec{q}_{i,j} \cdot w_{i,t} \cdot s_i + \sum_{i=m+1}^{m+p} \vec{q}_{i,j} \cdot w_{i,t} \cdot s_i \\
&= \sum_{i=1}^{m} \vec{q}_{i,j} \cdot w_{i,t} \cdot \epsilon^{m+p-i} + \sum_{i=m+1}^{m+p} \vec{q}_{i,j} \cdot w_{i,t} \cdot \epsilon^{m+p-i} \\
&= \epsilon^p \cdot \sum_{i=1}^{m} \vec{q}_{i,j} \cdot w_{i,t} \cdot \epsilon^{m-i} + \sum_{i=m+1}^{m+p} \vec{q}_{i,j} \cdot w_{i,t} \cdot \epsilon^{m+p-i} \\
&= \epsilon^p \cdot \vec{u}_{t,j} + \vec{u'}_{t,j}.
\end{aligned}$$

The deduction proves the additive property of the proposed personalized model U, so the proposed incremental update mechanism can be applied to maintain the model, rather than just recalculate the model based on all new and old records. Obviously, this process makes the personalized model computationally more efficient and promising.

4 EXPERIMENT RESULTS

In this section we discuss the conducted experiments to evaluate our proposed method and analyze the corresponding results. We first describe our experimental setup in Section 4.1. Then in Section 4.2, we find the proper parameters of the personalized model for the best facet-weight vector, and compare our proposed method with other existing methods.

4.1 Experimental Setup

We have implemented the proposed facet-weight based component ranking and personalized facet-weight calculation methods according to user's historical retrieval records, named *PWCRS* (Personalized Weight Based Component Retrieval System) (Microsoft .NET 2003 + SQL2000). We collect 2000 components from some public repositories such as Active-X [19], Alphaworks [20] etc. All of these components are classified by our facet classification scheme, which is defined based on Jade Bird Component Library System [21].

The experiments separate the users into three groups. Each group consists of 10 users. All the users know about the knowledge of component reuse to a certain extent, and what they should do is to search 10 components, whose abstract introductions have been shown to them. Users of group 1 take the traditional component retrieval method based on facet matching, which means all the facets have the same weights. Users of group 2 propose retrieval queries on each facet and set the weight of each facet themselves, using facet-weight based component retrieval method to search 10 target components. Users of group 3 are experienced in operating *PWCRS*, and each of them has used the system to retrieve more than 200 components. Our system has stored their historical retrieval records and constructed personalized models for each of them; so the users of group 3 will take the advantage of the automatically calculated facet-weights, i.e. the self-defined ones. If they are not satisfied, they can reset the facet-weight vectors.

There are three metrics to evaluate the efficiency of each group: *Average Retrieval Time (ART)*, *Average Target Position (ATP)*, and *Average Deviation (AD).*

Average Retrieval Time is defined as the average time of all users' whole retrieval process on searching one component, including the processes of user query requesting, component matching, and target component positioning in the resulting list. The statistics of *ART* among the three groups can be used to compare the time cost of the three methods. The shorter *ART* is, the higher the efficiency is.

Average Target Position is defined as the average position of target component in the resulting list among all users. For one target component, different user requirements and different retrieval methods will obviously result to different positions of the resulting list. The smaller *ATP* is, the more likely the target component is selected.

Average Deviation is defined as the average distance of difference between real weight vector and ideal weight vector of target component among all the users. The ideal weight vector means the one that leads to the maximal *GMD* of current user query $\vec{q_0}$ and target component $\vec{c}_{tar}$, which is $w_{ideal} = \arg\max(GMD(\vec{q_0}, \vec{c}_{tar}))$. The closer to the ideal weight vector, the better the target component is in the resulting order. The formula of *AD* can be defined as follows:

$$AD(\vec{w_0}, \vec{w}_{ideal}) = \sqrt{\sum_{i=1}^{n} (w_{0,i} - w_{ideal-i})^2}. \tag{7}$$

4.2 Results and Analysis

First, it is very important to figure out how the parameters influence the facet-weight in the personalized facet-weight calculation. There are two parameters: the number of historical retrieval records m and fading factor ϵ. Obviously, the change of these parameters will merely influence *ART*, because the time of component matching is independent with them. So, we just seek for the relationship between *ATP* and these two parameters, in order to figure out a suitable group of parameters m and ϵ for the best *ATP*. Users of group 3 directly achieve these tasks, and experimental results are shown as Figures 4 and 5. In Figure 4, the number of historical retrieval records m is statistically set as 120. In Figure 5, the fading factor ϵ is set to 0.95. The connected lines in both figures depict the average resulting values of the 10 users. The point above the average value indicates the maximal position of target component, while the point below the average value indicates the minimal position of target component.

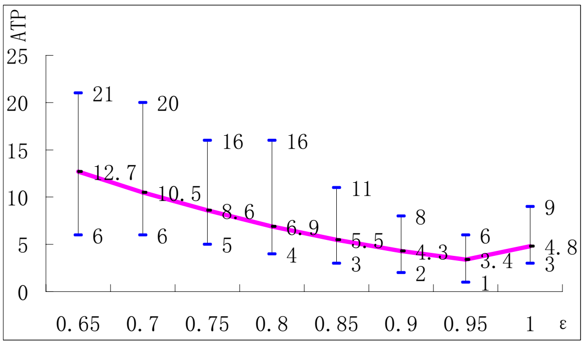

Fig. 4. The relationship of *ATP* and the fading factor

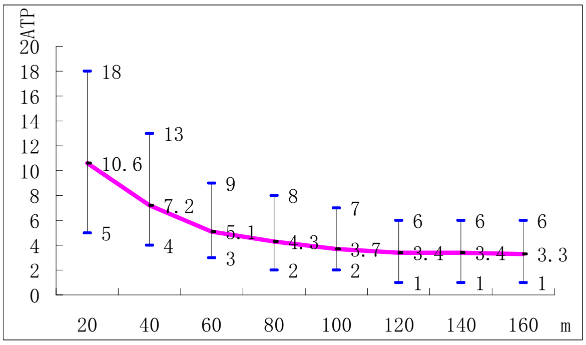

Fig. 5. The relationship of *ATP* and the number of records

Figure 4 shows that fading factor $\epsilon = 0.95$ makes the best calculation of facet-weight vector, which leads to the lowest *ATP* and all the target components are in top 10 of resulting lists. If the fading factor is set bigger or smaller, the target component will fall down in the resulting lists, much harder to be found out. In Figure 5, we can see that *ATP* goes smaller when number of historical retrieval records is set larger, and will be limited to a steady value when $m \geq 120$.

We then test and compare the results on AD among the three groups of traditional component retrieval method based on the facet matching, facet-weight based component retrieval method, and the personalized facet-weight based component retrieval method. Users of group 3 take the number of historical retrieval records $m = 120$ and the fading factor $\epsilon = 0.95$. The ideal weight vector $\vec{w}_{ideal}$ for current query $\vec{q}_0$ and target component $\vec{c}_{tar}$ can be calculated. The t^{th} element of $\vec{w}_{ideal}$ should be:

$$w_{ideal-t} = \frac{FMD(\vec{q}_{0,t}, \vec{c}_{tar-t})}{\sqrt{\sum_{i=1}^{n} FMD(\vec{q}_{0,i}, \vec{c}_{tar-i})^2}}. \tag{8}$$

Group 1 users do not use facet-weights, so the facet-weight vector in the corresponding experiment can be regarded as $(n^{-\frac{1}{2}}, n^{-\frac{1}{2}}, \ldots, n^{-\frac{1}{2}})$. The facet-weight vectors of group 2 users are defined by users themselves, while those in group 3 are calculated by the personalized model. According to the definition of AD in formula 7, the ADs of the three groups on each target component are shown in Figure 6.

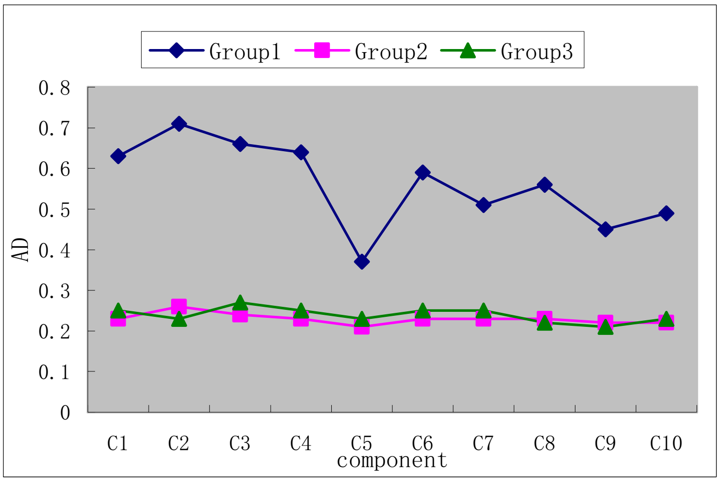

Fig. 6. Comparison of AD on different retrieval methods

The results shown in Figure 6 indicate that user defined facet-weight vectors are much closer to the ideal weight vector than the proportioned weights, and the calculated weights are nearly the same as what users really set. We can also see that the AD of group 2 and group 3 are very stable, while that of group 1 fluctuates much in different retrieval conditions. It proves that setting weights on facet brings positive support for improving the efficiency of component retrieval.

Finally, we compare the ART and ATP among three groups. Users of three groups search the same 10 components, regarded as $c_1, c_2, \ldots, c_{10}$. Users of group 3 still take the number of historical retrieval records $m = 120$ and the fading factor $\epsilon = 0.95$. The experimental results are shown in Figures 7 and 8.

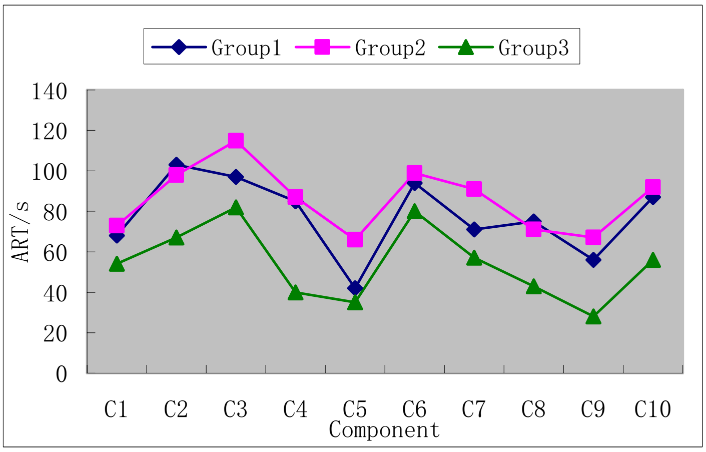

Fig. 7. Comparison of *ART* on different retrieval methods

The experimental results show that users of group 2 spend no more time than users of group 1 in the process of component retrieval. Obviously, compared with the retrieval operations of group 1, it takes extra time for user of group 2 to decide the weights for his query on all the facets; but the self-defined weights help users of group 2 optimize the component ranking and provide smaller *ATP* in the resulting list, which makes it easier for them to find out the target components, and saves time in component selection. So the total time of component retrieval process does not change significantly. The time users of group 3 spend is generally 30 % shorter than that of group 1 and group 2, because our system calculates facet-weight vectors after they set facet values on all facets. Users of group 3 are free to adjust the weights, but it hardly happens in our experiment. So this process takes much less time than that of the self-defined weights. Furthermore, the experiment of group 3 results in nearly the same *ATP* as those of group 2, which means the automatically calculated facet-weight vectors are very close to what the users really need. The experiment proves that the personalized model for facet-weight calculation indeed reflects the users search habits and interests, keeping high accuracy in component retrieval.

The comparison between the results of these three groups proves that setting facet-weight is quite useful and efficient in assigning differences of importance on all the facets, improving the accuracy of user requirement express, and helping users select satisfying components quickly. User's historical operations of component

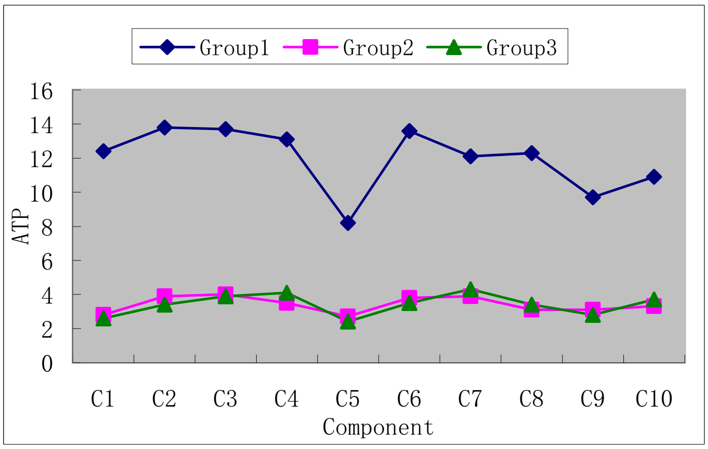

Fig. 8. Comparison of *ATP* on different retrieval methods

retrieval on setting value and weight on each facet reflect his common habits and interests, and the importance of this relationship will attenuate over time; so the personalized model constructed according to these characters is able to automatically calculate reasonable facet-weight vectors for user's current queries, so as to save user's time and improve the retrieval efficiency.

5 CONCLUSIONS AND FUTURE WORK

The emerging ubiquitous computing is changing the current computing paradigm and our daily activities. In order to make the ubiquitous services accessible to anyone, anytime and anywhere, it is important to improve the retrieval efficiency of current component resources. A good ranking method helps reduce the time spending to accurately select suitable components, which is of great foundation for providing ubiquitous services. In this paper we have proposed a personalized facet-weight based ranking method for component retrieval. We designed a new formula to rank the facet-based components by assigning a weight on each facet to distinguish the importance of facet values, and automatically calculate facet-weights for user according to his historical retrieval records of facet values and weight setting. To reduce the calculation of facet-weight calculation, we created a personalized model to integrate user's historical records of facet values and weights, fading factor, and deduct the formula of incremental study to append user's new retrieval records to the personalized model. The experimental results optimize the parameters of personalized model for facet-weight calculation, and prove that the personalized facet-weight based component ranking method performs better than the traditional facet-based component retrieval method in terms of retrieval accuracy and execution time.

In the future we plan to expand the ranking scale to heterogeneous components repositories. In the environment of ubiquitous computing, it often happens that users need to organize heterogeneous components to ensure completing their required service functions. Ranking among more components repositories may help retrieve better components for user requirements. We also plan to take more user-specific information into consideration of the personalized model, and to design more sophisticated learning and ranking algorithms to further improve the efficiency of component retrieval.

REFERENCES

[1] Zhang, Y. X.—Fang, C. H.: Active Service: Concept, Architecture and Implementation. USA: Thomson Learning, 2005.

[2] Wolfgang, P.: Component-Based Software Development – A New Paradigm in Software Engineering? In Proceedings of the Fourth Asia-Pacific Software Engineering and International Computer Science Conference 1997, p. 523.

[3] Frakes, W. B.—Pole, T. P.: An Empirical Study of Representation Methods for Reusable Software Components. IEEE Transactions on Software Engineering. Vol. 120, 1994, No. 8, pp. 617–630.

[4] Mili, H.—Rada, R.—Wang, W. et al.: A Comparative Study of Two Software Reuse Research Projects., Journal of Systems and Software, Vol. 25, 1994, No. 2, pp. 147–170.

[5] London, R. L.: Specifying Reusable Components Using Z: Realistic Sets and Dictionaries. In Proceedings of Fifth International Workshop on Software Specification and Design, ACM SIGSOFT Software Engineering Notes, Vol. 14, May 1989, No. 3, pp. 120–127.

[6] Zarenski, A. M.—Jeannette, W. M.: Specification Matching of Software Components. In: Proceedings of the ACM SIGSOFT '95 Symposium on Foundations of Software Engineering, Vol. 20, October 1995, No. 4, pp. 6–17.

[7] Wang, S. F.—Wang, K. H.: Research on the Knowledge-Based Reusable Component Retrieval System KRR. Computer Engineering and Applications. Vol. 36, March 2000, No. 3, pp. 1–3.

[8] Tomas, I.—Robert, J. K.: Supporting Search for Reusable Software Objects. IEEE Transactions on Software Engineering, Vol. 22, June 1996, No. 6, pp. 407–423.

[9] Diaz, R. P.—Freeman, P.: Classifying software for reusability. IEEE Software 1987, Vol. 4, No. 1, pp. 6–16.

[10] NEC Software Engineering Laboratory. NATO Standard for Management of a Reusable Software Component Library. NATO Communications and Information Systems Agency, Tokyo, Japan, Vol. 2, 1991, pp. 32–43.

[11] Morel, J. M.—Faget, J.: The REBOOT Environment. In Proceedings of the 2nd InternationalWorkshop on Software Reusability Advances in Software, Lucca: IEEE Computer Society Press, 1993, pp. 80–88.

[12] Thorsten, R.: A New Measure of the Distance between Ordered Trees and Its Applications. Research Report, 85166, Department of Computer Science, University of Bonn, Germany 1997.

[13] Torshen, S.—Naumann, F.: Approximate Tree Embedding for Querying XML Data. In Proceedings of ACM SIGIR Workshop on XML and Information Retrieval, Athens, Greece 2000.

[14] Shasha, D.—Tsong, J.—Wang, L.: Exact and Approximate Algorithm for Unordered Tree Matching. IEEE Transactions on Systems, Man and Cybernetics, Vol. 24, 1994, No. 4, pp. 668–678.

[15] Gibb, K.—McCartan, C.—O'Donnell, R.—Sweeney, N.—Leon, R.: The Integration of Information Retrieval Techniques within a Software Reuse Environment. Journal of Information Science, Vol. 26, 2000, No. 4, pp. 520–539.

[16] Wang, Y. F.—Xue, Y. J.—Zhang, Y.—Zhu, S. Y.—Qian, L. Q.: A Matching Model for Software Component Classified in Faceted Scheme. Journal of Software. Vol. 14, 2003, No. 3, pp. 401–408.

[17] Yang, Y.—Zhang, W. S.—Zhang, X. G.—Shi, J. Y.: A Weighted Ranking Algorithm for Facet-Based Component Retrieval System. In Proceeding of 2nd IASTED International Conference on Advances in Computer Science and Technology, Puerto Vallarta, Mexico 2006, pp. 274–279.

[18] Xie, X. Q.—Tang, J.—Li, J. Z.—Wang, K. H.: A Component Retrieval Method Based on Facet-Weight Self-Learning. In Proceeding of 2004 Advanced Workshop on Content Computing (AWCC), LNCS 3309, 2004, pp. 437–448.

[19] `http://www.active-x.com`.

[20] `http://www.alphaworks.ibm.com`.

[21] Chang, J. C.—Li, K. Q.—Guo, L. F.—Mei, H.—Yang, F. Q.: Representing and Retrieving Reusable Software Components in JB (Jadebird) System. Electronica Journal, Vol. 28, 2000, No. 8, pp. 20–24.

Ming Zhong received the B. Sc. degree in Computer Science and Technology from Tsinghua University, Beijing, China in 2005. At present, he is a Ph. D. candidate in Department of Computer Science and Technology in Tsinghua University. His research interests are in the area of program mining, component organizing, and searching. He has been involved in various projects on Component Based Software Development.

Yaoxue Zhang received his B. Sc. degree from Northwest Institute of Telecommunication Engineering, China, and received his Ph. D. degree in Computer Networking from Tohoku University, Japan in 1989. Then he joined Department of Computer Science, Tsinghua University, China. He was a Visiting Professor at Massachusetts Institute of Technology (MIT) and University of Aizu in 1995 and 1998, respectively. Currently, he is a fellow of the Chinese Academy of Engineering and a Professor in computer science and technology at Tsinghua University, China. He also serves as an editorial board member of four international journals. His major research areas include computer networking, operating systems, ubiquitous/pervasive computing, transparent computing, and active services. He has published over 170 technical papers in international journals and conferences, as well as 8 monographs and textbooks.

Laurence Tianruo Yang focuses on the research fields of high performance computing and networking, embedded systems, ubiquitous/pervasive computing and intelligence. He has published about 300 papers (including about 80 international journal papers such as IEEE and ACM Transactions) in refereed journals, conference proceedings, and book chapters in these areas. He has been involved in more than 100 conferences and workshops as a program/general/steering conference chair and in more than 300 conference and workshops as a program committee member. He served as the Vice-Chair of IEEE Technical Committee of Supercomputing Applications (TCSA) until 2004; currently he is the Chair of IEEE Technical Committee of Scalable Computing (TCSC), the Chair of IEEE Task force on Ubiquitous Computing and Intelligence. In addition, he is the editor-in-chief of eight international journals and several book series. He is also serving as an editor for about 20 international journals.

Yuezhi Zhou received his Ph. D. degree in Computer Science and Technology from Tsinghua University, China in 2004 and is now working as an Associate Professor at the same university. He worked as a Visiting Scientist at the Computer Science Department in Carnegie Mellon University in 2005. His research interests include ubiquitous/pervasive computing, distributed systems, mobile devices and systems. He has published over 30 technical papers in international journals or conferences. He received the IEEE Best Paper Award in the 21st IEEE AINA International Conference in 2007. He is a member of IEEE and ACM.

Pengwei Tian received his B. Sc. degree in Computer Science and Technology from North-Eastern University, Shenyang, China in July 2004. From September 2004 till now, he has been a Ph. D. student in Department of Computer Science and Technology, Tsinghua University, Beijing, China. His research interests focus on software service reuse, service evolution, active service modeling, and QoS issues in the ubiquitous computing environment. He is the co-author of 10 technical papers and 4 Chinese patents.

Linkai Weng has received his B. Sc. degree in Computer Science and Technology from Tsinghua University, Beijing, China in 2006. Then he became a Ph. D. student in the same University. His main research interest focuses on personalized recommendation in social community.